Embedded Linux Conference Europe

15th October 2014 - Dusseldorf, Germany

Tame the USB gadgets talkative beast

Krzysztof Opasiak,
k.opasiak@samsung.com
Samsung R&D Institute Poland

Agenda

- USB Overview
- ConfigFS composite gadget
- libusbg & gt
  - C API
  - Gadget Schemes
  - gt
- gadgetd & gadgetctl
  - Features
  - gadgetctl
- Q&A

USB Overview

Host vs Devices

Host *is being extended with some functionality by device*

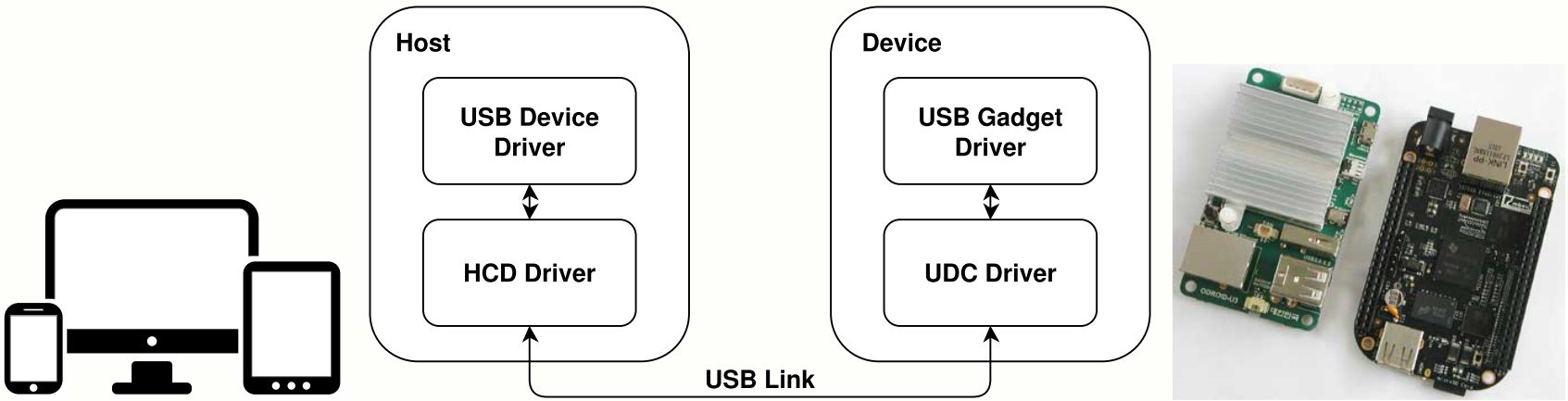
HCD driver - Driver for Host Controller

USB device driver - Driver for particular USB functionality/device

Device *extends the host with some functions*

UDC driver Driver for USB Device Controler

USB Gadget driver Driver implementing peripheral logic

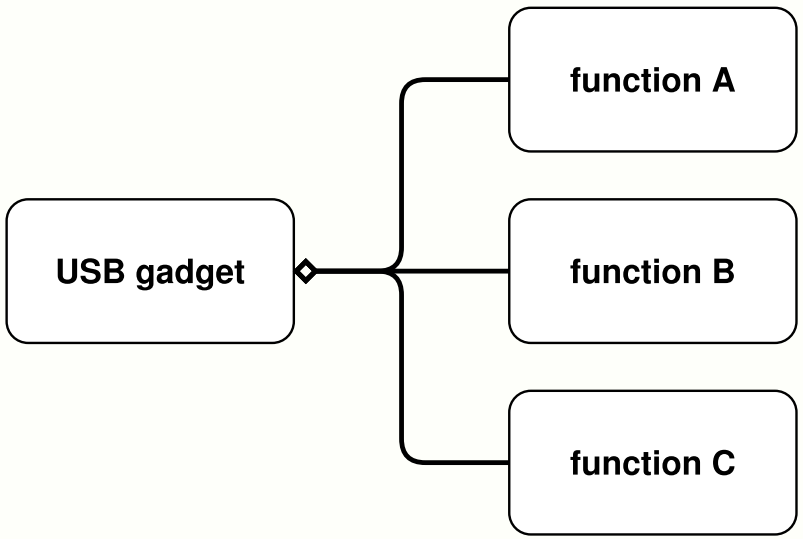


USB Composite Device

Functions

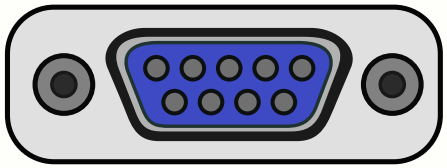
USB Function

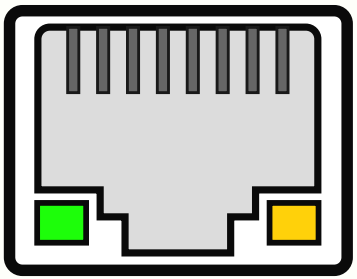
- **Independent from each other**
- **Set of USB interfaces**
- **Implementation of some protocol (ex. HID, Mass Storage or Custom)**
- **Piece of code in kernel module**



USB functions in kernel

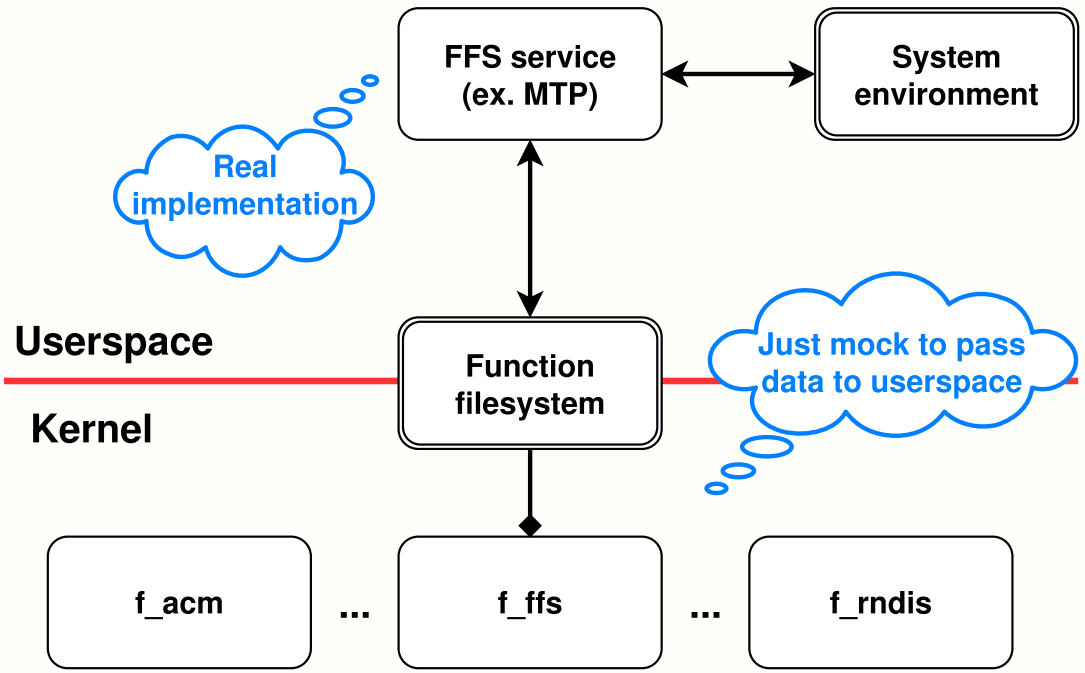
- **Serial**
  - ACM
  - Serial
  - OBEX
- **Ethernet**
  - ECM
  - EEM
  - NCM
  - Subset
  - RNDIS
- **Phonet**
- **Mass Storage**
- **Loopback**
- **SourceSink**
- **UVC and HID - WIP**





FunctionFS

- **Access to system environment**
- **Easier implementation**
- **Kernel USB function & file system**
- **Wraps file IO operations into usb_requests**



FunctionFS - HOWTO

- **Usage:**
  - open ep0 file
  - Write function descriptors
  - Write function strings
  - Open epXX (if any)
  - Read events from ep0
  - …(Protocol specific)
- **Performance - use Async IO**
- **See Alan's Ott presentation:**

"USB and the Real World"

Gadget composition

- **Fill the identity of gadget**
  - Vendor ID
  - Product ID
  - Device Class details
  - Strings (manufacturer, product and serial)
- **Decide what functions it has**
- **Decide how many configurations**
- **Decide what functions are available in each configuration**

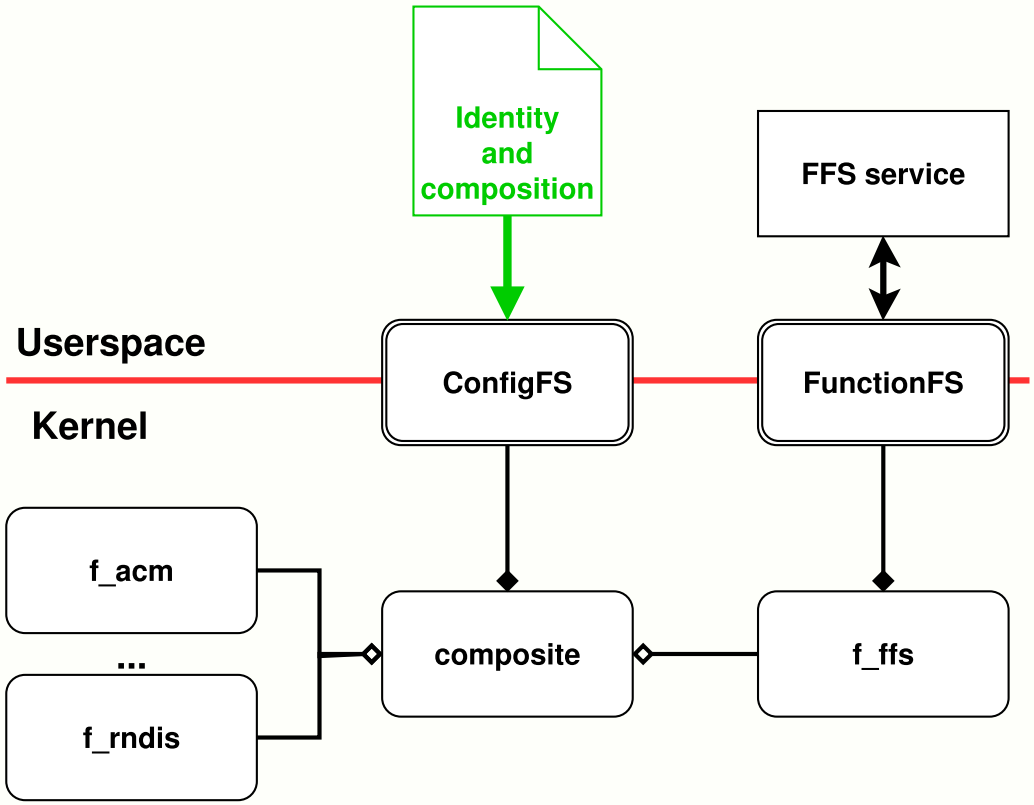
Gadget composition: old-school

- **One gadget one kernel module**
- **Composition and identity hardcoded in .c file**
- **Module parameters**
  the only way of gadget modification
- **One gadget build into kernel or**
  a few as modules
- **Small change - recompilation**
- **Simple usage:**

```
$ modprobe g_ether
```

Gadget composition: ConfigFS

- **Allow user to compose gadget at runtime**
- **Register as subsystem in ConfigFS**
- **Use file system ops to compose a gadget**
- **Load module on request**



ConfigFS composite gadget

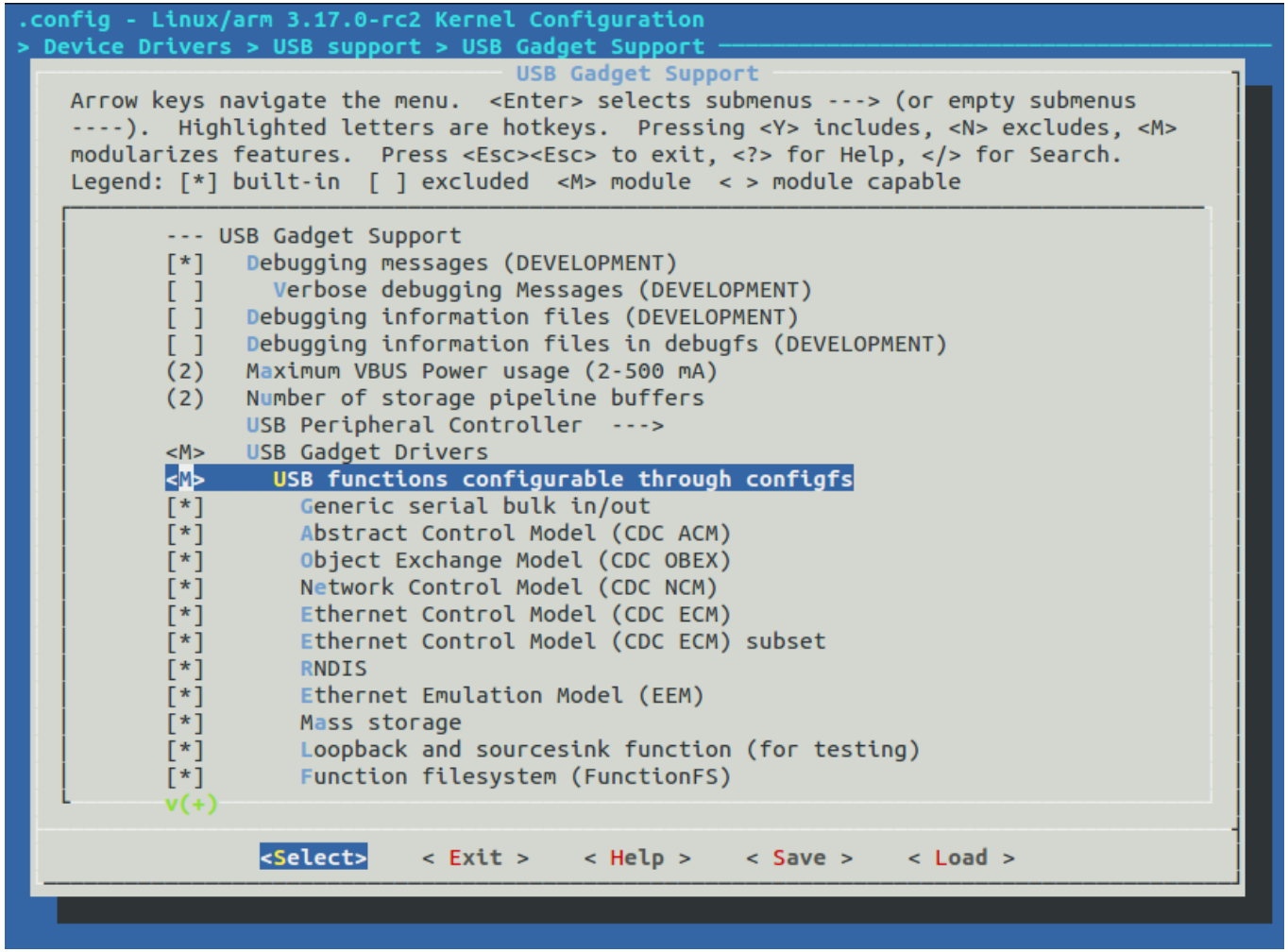
- **Separate code from configuration**
- **New composition without recompilation**
- **Provide blocks and framework to compose them**
- **Flexibility**

ConfigFS composite gadget

- **Separate code from configuration**
- **New composition without recompilation**
- **Provide blocks and framework to compose them**
- **Flexibility**
- **Usage:**

ConfigFS composite gadget

Prerequisites - menuconfig



Give a chance to request_module() - use depmod

Prologue

Prologue

```
$ modprobe libcomposite
$ mount none -t configfs /sys/kernel/config
$ cd /sys/kernel/config/usb_gadget
```

Simple example

Create gadget, fill identity

```
$ mkdir g1
$ cd g1
$ echo "0x1d6b" > idVendor
$ echo "0x0104" > idProduct
$ mkdir strings/0x409
$ echo "My␣serial" > strings/0x409/serialnumber
$ echo "My␣Vendor" > strings/0x409/manufacturer
$ echo "My␣Product" > strings/0x409/product
```

Simple example

One config, one function

```
$ mkdir functions/rndis.usb0
$ mkdir configs/c.1
$ mkdir configs/c.1/strings/0x409
$ echo "Config␣1" > \
       configs/c.1/strings/0x409/configuration
$ ln -s functions/rndis.usb0 configs/c.1/
```

Simple example

Check UDC name

```
$ ls /sys/class/udc
12480000.hsotg
```

Bind gadget to udc

```
$ echo "12480000.hsotg" > UDC
```

Simple example

On host side

```
$ lsusb -v
Bus 003 Device 026: ID 1d6b:0104
  bcdUSB               0.00
  bDeviceClass            0
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x1d6b Linux Foundation
  idProduct          0x0104 Multifunction Composite
  bcdDevice            3.17
  iManufacturer           1 My Vendor
  iProduct                2 My Product
  iSerial                 3 My serial
  bNumConfigurations      1
```

ConfigFS and FunctionFS

ConfigFS modifications

```
$ echo "" > UDC
$ mkdir functions/ffs.my_func_name
$ ln -s functions/ffs.my_func_name configs/c.1/
$ mount my_func_name -t functionfs /tmp/mount_point
$ run_function_daemon
$ wait_for_daemon_initialization
$ echo "12480000.hsotg" > UDC
```

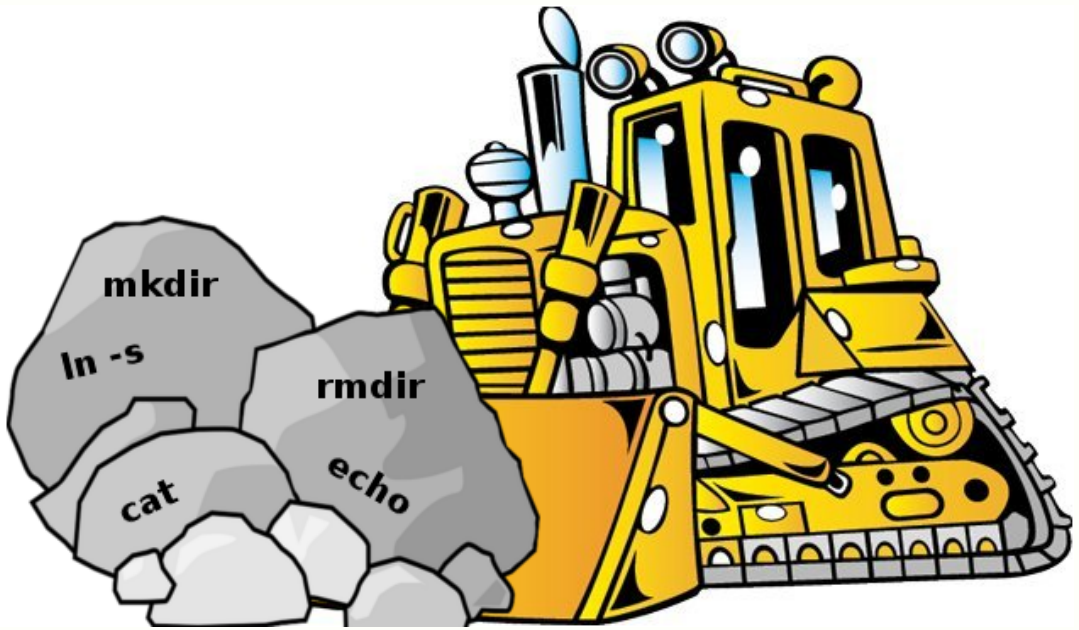
Problems?

- **Very verbose (~20 commands for simple gadget)**
- **A lot of dependencies, magic numbers and syntax rules**
- **User has to know function types offered by current kernel**
- **Generate unique instance names for FFS**
- **Run demon manually and pass mount point**
- **Complex userspace function setup**
- **Reliability - death of daemon causes gadget unbind**
- **Limited security - only unix users rights**

libusbg & gt

libusbg - goals

- **Allow to create gadget from code**
- **Provide abstraction layer for ConfigFS**
- **Reduce number of magic numbers**
- **Limit number of potential mistakes**
- **Allow for fast and easy gadget creation**
- **Make gadget creation declarative**



Few words about libusbg

- **Announced by Matt Porter in September 2013**
- **C library for fast and easy gadget creation**
- **Official:**

  `https://github.com/libusbg/libusbg`

- **Unofficial (my devel):**

  `https://github.com/kopasiak/libusbg`

- **Review through linux-usb and pull requests**
- **Support for almost all USB functions**
- **Just take the code and check it!**

C API Overview

- **Opaque structures for all entities:**
  - usbg_state
  - usbg_gadget
  - usbg_config
  - usbg_function
  - usbg_binding
  - usbg_udc
- **usbg_gadget_attrs - gadget attributes, similar layout to libusb_device_descriptor ;)**
- **No static buffers, reentrant**
- **Snapshot taken on initialization**

Example

Attributes and strings

```
static usbg_gadget_attrs g_attrs = {
  /* Class defined at interface level */
  .idVendor = 0x1d6b,
  .idProduct = 0x104,
};

usbg_gadget_strs g_strs = {
  .str_ser = "My␣serial",
  .str_mnf = "My␣Vendor.",
  .str_prd = "My␣Product␣Name",
};

usbg_config_strs c_str = {
  "Config␣1"
};
```

Example

Gadget creation

```
usbg_init("/sys/kernel/config", &s);

usbg_create_gadget(s, "g1", &g_attrs, &g_strs, &g);
usbg_create_function(g, F_RNDIS, "usb0", NULL, &f_rndis);
usbg_create_config(g, 1, "c", NULL, &c1_strs, &c);

usbg_add_config_function(c1, "rndis_func", f_rndis);

usbg_enable_gadget(g, DEFAULT_UDC);
usbg_cleanup(s);
```

Checkpoint

- **Flexible API for gadget creation**
- **Fast gadget removal**
  - Hardcode in shell script
  - Hardcode in C program
  - Maybe some cmd line tool?
- **Still no equivalent to**

```
$ modprobe g_ether
```

Gadget schemes

- **Use configuration files for gadget composition**
- **libconfig syntax instead of reinventing the wheel**
- **Set of usbg_import_*() and usbg_export_*() functions**
- **Finally close to** *modprobe g_ether* **!!!**

Example

Canonical form

```
attrs = {
 idVendor = 0x1D6B
 idProduct = 0x104
}

strings = ({
  lang = 0x409;
  manufacturer = "My␣vendor"
  product = "My␣product"
  serialnumber = "My␣Serial"
})
```

Example

Canonical form

```
functions = {
  rndis_func = {
    instance = "usb0"
    type = "rndis"
  }
}

configs = ({
  id = 1
  name = "c"
  strings = ({
    lang = 0x409
    configuration = "Config␣1"
  })
  functions = ("rndis_func")
})
```

Example

Shorter form

```
attrs = {idVendor = 0x1D6B; idProduct = 0x104;}

strings = ({
  lang = 0x409;
  manufacturer = "My␣vendor"
  product = "My␣product"
  serialnumber = "My␣Serial"
})

configs = ({
  id = 1
  name = "c"
  strings = ({lang = 0x409; configuration = "Config␣1";})
  functions = ({function = {instance = "usb0"; type = "rndis";}})
})
```

Example

Gadget loader

```
usbg_init("/sys/kernel/config", &s);

file = fopen("my_gadget.gs", "r");

usbg_import_gadget(s, file, "g1", &g);

usbg_enable_gadget(g, DEFAULT_UDC);
usbg_cleanup(s);
```

Example

Gadget loader

```
usbg_init("/sys/kernel/config", &s);

file = fopen("my_gadget.gs", "r");

usbg_import_gadget(s, file, "g1", &g);

usbg_enable_gadget(g, DEFAULT_UDC);
usbg_cleanup(s);
```

- **More gadget schemes tweaks:**

  https://github.com/kopasiak/libusbg/tree/master/doc

- **More examples:**

  https://github.com/kopasiak/libusbg/tree/master/examples

libusbg - our plans

- **Support for all other USB functions**
- **Support for OS descriptors**
- **API improvements**
- **Schemes with equivalents of legacy gadgets**
- **Multi-process awareness**
- **Change notifications**
- **Tests**

Gadget tool

- **C API is not enough**
- **Access to libusbg goodies from command line**
- **Easy gadget administration**
- **Combine libusbg examples into one binary**
- **Finally gadget composed with one command!**

gt

- Command line tool for gadget management
- Uses libusbg
- Developed on github:

  https://github.com/kopasiak/gt
- WIP - initial state
- First few commands working
- Significant contribution from Paweł Szewczyk

UNDER CONSTRUCTION

Example

One config, one function

```
$ gt create g1 \
    idVendor=0x1d6b \
    idProduct=0x104 \
    manufacturer="My␣Vendor" \
    serialnumber="My␣Serial" \
    product="My␣product"

$ gt func create g1 rndis usb0
$ gt config create g1 label 1
$ gt config add g1 1 rndis usb0
$ gt enable g1
```

Checkpoint

- **Very verbose (~20 commands for simple gadget)**
- **A lot of dependencies, magic numbers and forms**

SOLVED

- **User has to know function types offered by current kernel**

PARTIALLY SOLVED

- **Generate unique instance names for FFS**
- **Run demon manually and pass mount point**
- **Complex userspace function setup**
- **Reliability - death of daemon causes gadget unbind**
- **Limited security - only unix users rights**

NOT SOLVED

gadgetd & gadgetctl

gadgetd - goals

- **System-wide USB gadget management**
- **Uniform API for kernel and userspace functions**
- **High level API**
- **Simplify userspace functions**
- **Resource efficiency**
- **Extend Security**
- **Make USB gadgets easy to use**

Few words about gadgetd

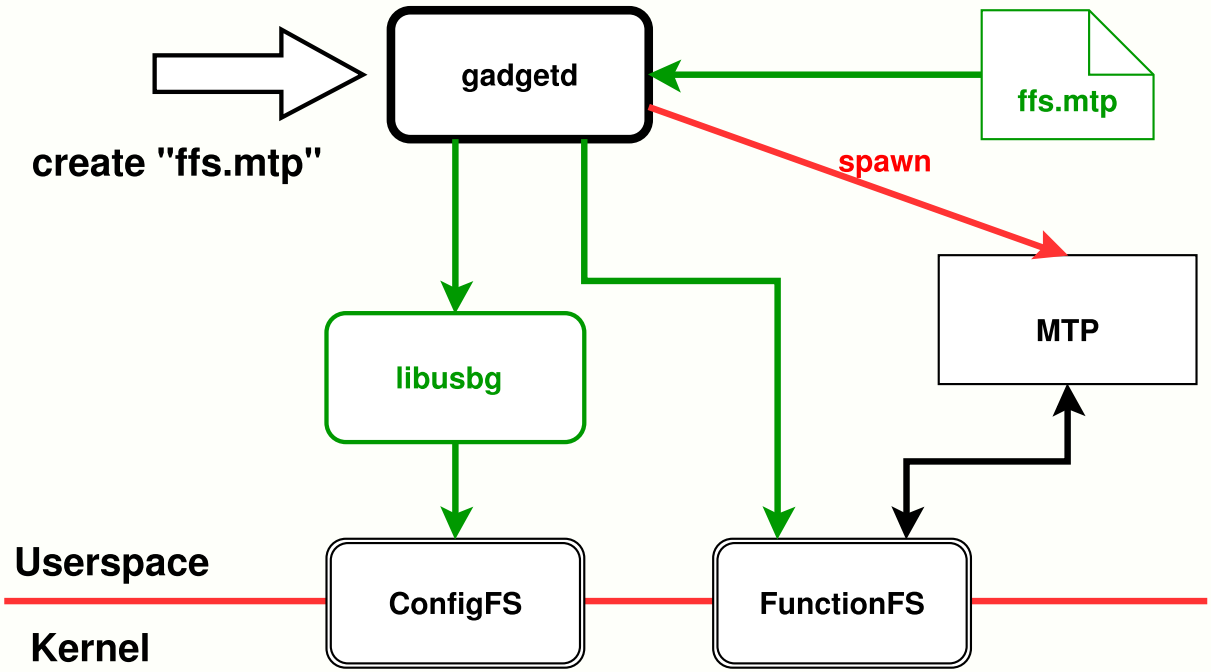
- **Created by me and Stanisław Wadas**
- **Developed on github:**
  `https://github.com/gadgetd/gadgetd`
- **Idea in early 2014**
- **Uses libusbg**
- **WIP**
- **Proof of Concept ready**

gadgetd - features

- **Userspace and kernel functions unification**
- **Lazy user functions startup**
- **Functions introspection**
- **Polkit/Cynara cooperation**
- **DBUS API**

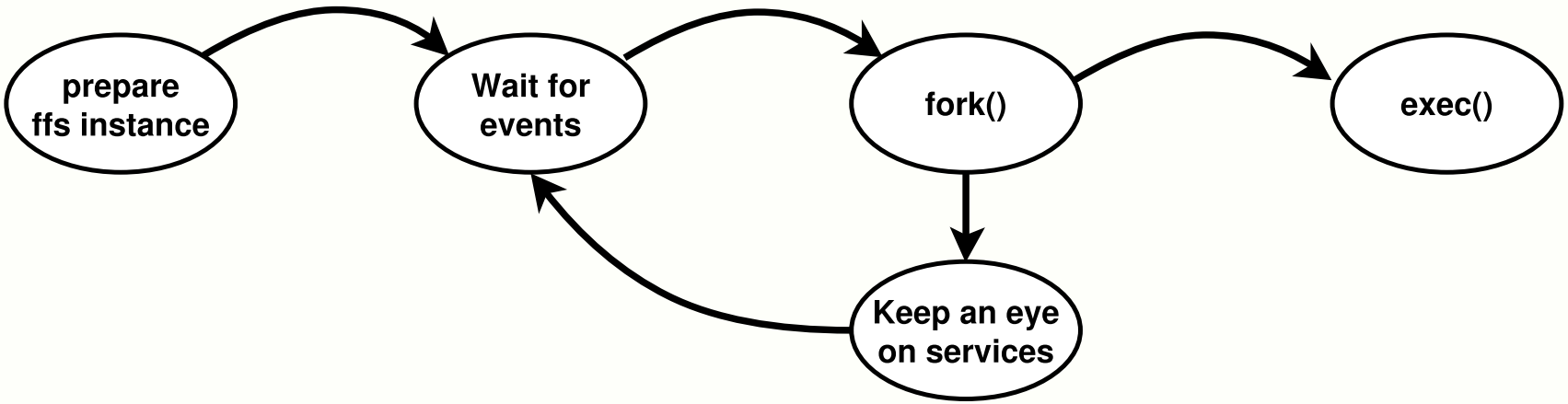
Functions unification

- Separate code and configuration
- Provide config file for each user function
- New function type for each service
- Take care of naming and mounting
- User simply creates "ffs.mtp" function
- Handle instance naming issues



FFS-inet

- Only enabled functions are required
- Just In Time service startup
- Wait for events on all ep0
- Spawn on enable, pass descriptors
- Keep an eye on services
- Error handling policy



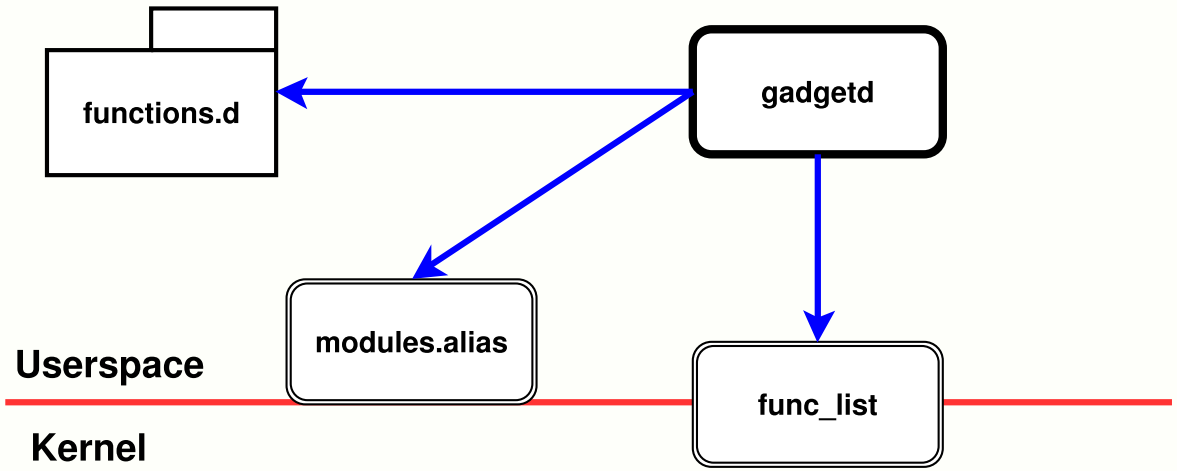
Introspection

- User need to know function types
- How to collect all type names?
- Check config files for user functions
- Check available modules
- How to reliably check build-in functions?

Currently IMPOSSIBLE!!!

We are working on it:

http://article.gmane.org/gmane.linux.usb.general/111068



Security - TODO

- **USB profiles**
  charger, private, public…
- **USB gadget for everyone or no?**
- **Different functions for different users?**
- **This is definitely policy!**
- **What to do?**
- **Ask someone who knows:**

Polkit || Cynara

DBUS API

- **High-level interface**
- **USB entities as DBUS objects**
- **Specialized interfaces for functions**
- **Good for configuration purposes**

gadgetctl

- Command line tool for gadgetd
- Reference DBUS client
- Share the code with gt
- Developed in the same repo as gt
- Second backend for gt
- WIP - very initial state

Example

One config, one ffs function

```
$ gadgetctl create g1 \
    idVendor=0x1d6b \
    idProduct=0x104 \
    manufacturer="My␣Vendor" \
    serialnumber="My␣Serial" \
    product="My␣product"

$ gadgetctl func create g1 ffs.sample i_name
$ gadgetctl config create g1 label 1
$ gadgetctl config add g1 1 ffs.sample i_name
$ gadgetctl enable g1
```

gadgetd - future work

- **Code cleanup**
- **Continue API implementation**
- **Simplify service config files format**
- **Finish service lifecycle management**
- **Improve ffs-daemon library implementation**
- **Tests**

Summary

- Very verbose (~20 commands for simple gadget)
- A lot of dependencies, magic numbers and forms
- User has to know function types offered by current kernel
- Generate unique instance names for FFS
- Run demon manually and pass mount point
- Complex userspace function setup
- Reliability - death of daemon causes gadget unbind
- Limited security - only unix users rights

SOLVED

Q&A

Questions?

Don't ask how to use USB gadget, ask how to develop it!

k.opasiak@samsung.com

Refereces

- **Andrzej Pietrasiewicz, Make your own USB gadget**
- **Matt Porter, Kernel USB Gadget ConfigFS Interface**
- https://github.com/gadgetd/gadgetd/wiki
- https://github.com/libusbg/libusbg
- https://github.com/kopasiak/libusbg
- https://github.com/kopasiak/gt
- https://github.com/hyperrealm/libconfig
- http://lwn.net/Articles/395712/
- https://wiki.tizen.org/wiki/USB/Linux_USB_Layers/Configfs_Composite_Gadget
- https://wiki.tizen.org/wiki/Security:Cynara
- http://www.freedesktop.org/wiki/Software/polkit/